EECS 280 - Lecture 21

Auto, Maps, Range-Based For Loops

Reminders/Updates

Check EECS280.org for updates

Project 5 is due Monday June 20th at 8pm

-BST and Map done by Tuesday June 13th

-main.cpp done by Monday June 20th

Final Exam - Thursday June 23 4-6pm

Lab Tomorrow!

Optional Special Topics Lecture Monday June 20th

AI, machine learning, and cognitive science!

Agenda

Functors Review

Containers Review

- -BSTs
- -Sets
- -Maps

C++ Shortcuts

Recursion Practice

Review

Classes - we define our own type and its behavior

Iterator - a class that acts like a pointer

-overloaded `*, ++, !=`

Functors - a class that acts like a function

-overloaded `( )`

Types of Functions/Functors

Predicate - takes an input and returns a bool

ex) isEven, isNegative

Comparator - Takes in two objects and returns true if the first object is less than the second object

ex) less

Functor Exercise

https://onlinegdb.com/pg0gaCUON

Write a functor that prints out an element to cout that can be used with this `for_each` function.

```
template <typename Iter_t, typename Func_t>

Func_t for_each(Iter_t begin, Iter_t end, Func_t func) {

  for (Iter_t it = begin; it != end; ++it) {

    func(*it);

  }

  return func;

}
```

Printer Functor Solution

```
template <typename T>

class Printer {

public:

    void operator()(const T &n) const {

        cout << n;

    }

};
```

```
int main(){

    list<int> lis; //fill with numbers



    for_each(lis.begin(), lis.end(),
Printer<int>());

}
```

How to modify Printer so it can use any stream?

```
template <typename T>

class Printer {



public:



    void operator()(const T &n) const {

        cout << n;

    }

};
```

Hint: Which of the following should you add?

a. Member variable
b. constructor
c. Destructor
d. operator*
e. Multiple of above

How to modify Printer so it can use any stream?

```
template <typename T>
class Printer {
    ostream &os;
public:
    Printer(ostream &os_in) : os(os_in) {}
    void operator()(const T &n) const {
        os << n;
    }
};
```

Hint: Which of the following should you add?

a. Member variable
b. constructor
c. Destructor
d. operator*
e. Multiple of above

How to modify Printer so it can use any stream?

```
template <typename T>

class Printer {

     ostream &os;

public:

     Printer(ostream &os_in) : os(os_in)
{}

     void operator()(const T &n) const {
os << n; }

};
```

```
int main(){

     list<int> lis; //fill with numbers

     ofstream fout("list.out");

     for_each(lis.begin(), lis.end(),
Printer<int>(fout));

}
```

Agenda

Functors Review

Containers Review

- **BSTs**
- Sets
- Maps

C++ Shortcuts

Recursion Practice

Review: Containers - holds elements

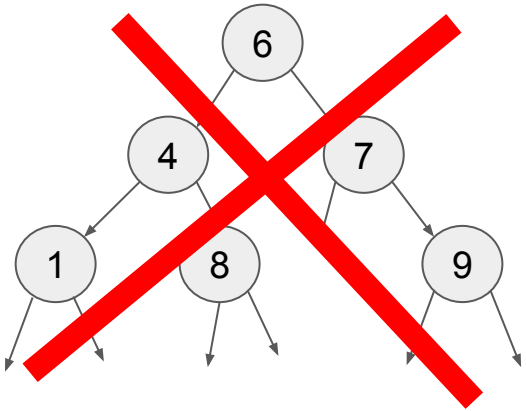
- Arrays
- Vectors
- Unordered_Set
- Sorted_Set
- List
- **Binary Search Tree (BST)**

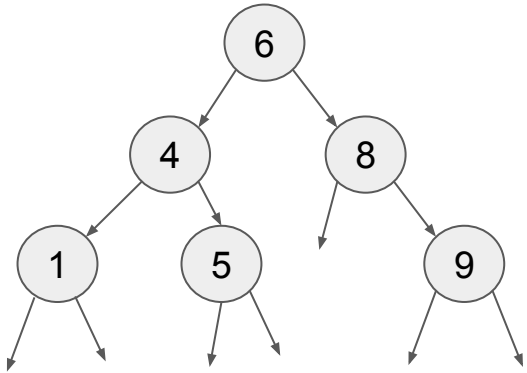
… many more!

Has things like:
- Insert (push_front, push_back…)
- Contains
- Size
- Remove
- Iterators

Review: Binary Search Trees (BSTs)

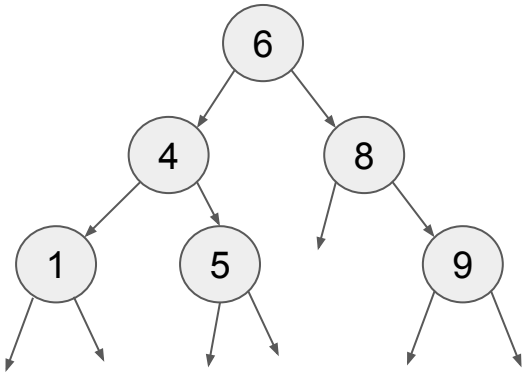
- A tree whose subtrees are also binary search trees
- Every element in the left subtree is strictly less than the root datum
- Every element in the right subtree is strictly greater than the root datum





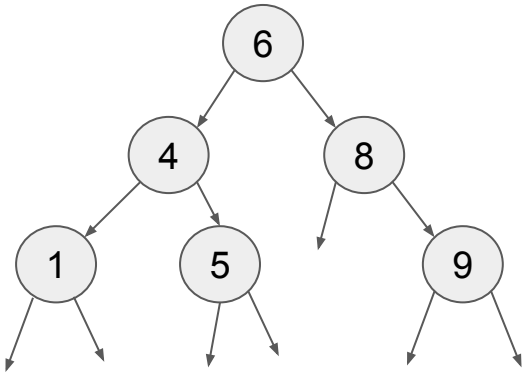
Binary Search Tree Interface

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```



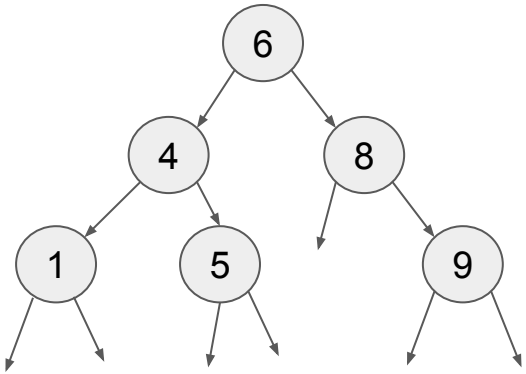
Binary Search Tree Interface

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```



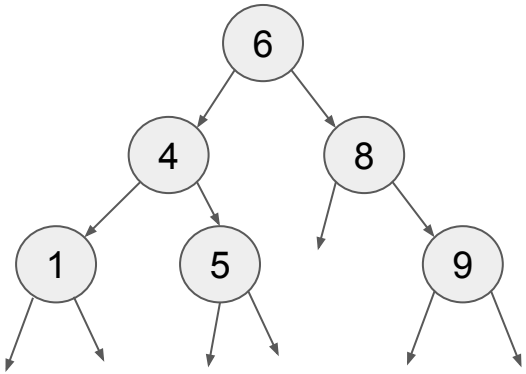
Binary Search Tree Interface

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```



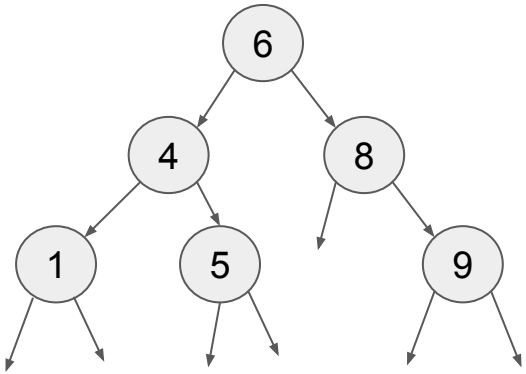
Binary Search Tree Interface

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```



Binary Search Tree Interface

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```



Agenda

Functors Review

Containers Review

- -BSTs
- **-Sets**
- -Maps

C++ Shortcuts

Recursion Practice

Review: Containers - holds elements

- Arrays
- Vectors
- **Unordered_Set**
- **Sorted_Set**
- List
- Binary Search Tree (BST)

… many more!

Has things like:
- Insert (push_front, push_back…)
- Contains
- Size
- Remove
- Iterators

Partner: What is a set?

Review: Containers - holds elements

- Arrays
- Vectors
- **Unordered_Set**
- **Sorted_Set**
- List
- Binary Search Tree (BST)

… many more!

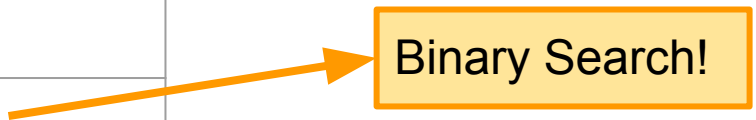
Has things like:
- Insert (push_front, push_back…)
- Contains
- Size
- Remove
- Iterators

Partner: What is a set?

Set - holds only unique items

Set Efficiency

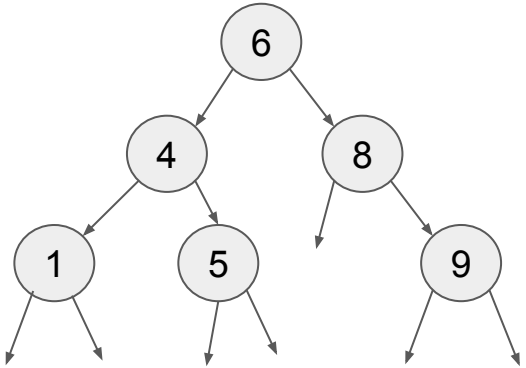
Function	Unsorted_Set	Sorted_Set
`size`	O(1)	O(1)
`contains`	O(n)	O(log n)
`insert`	O(n)	O(n)



Set Efficiency

A Set using a BST under the hood?

Function	Unsorted_Set	Sorted_Set	BST_Set
`size`	O(1)	O(1)	
`contains`	O(n)	O(log n)	
`insert`	O(n)	O(n)	



Set using a BST

```
template <typename T>
class BST_Set {
public:
    bool contains(const T &v) const { return elts.contains(v); }
    void insert(const T &v){ if(!contains(v)) elts.insert(); }
    int size() const { return elts.size(); }

private:
    BinarySearchTree<T> elts;
};
```

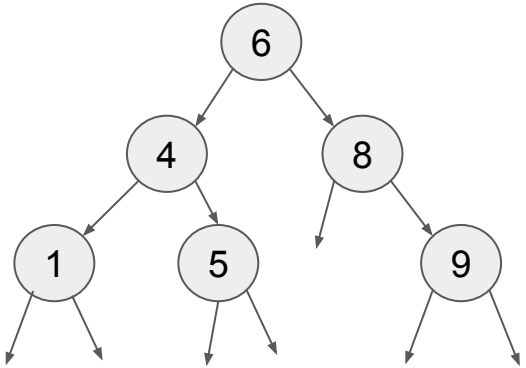
Set using a BST

```
template <typename T>

class BST_Set {

public:

    bool contains(const T &v) const { return elts.contains(v); }

    void insert(const T &v){ if(!contains(v)) elts.insert(); }

    int size() const { return elts.size(); }


private:

    BinarySearchTree<T> elts;

};
```

Set Efficiency

A Set using a BST under the hood?

Function	Unsorted_Set	Sorted_Set	BST_Set
`size`	O(1)	O(1)	
`contains`	O(n)	O(log n)	O(log n)
`insert`	O(n)	O(n)	



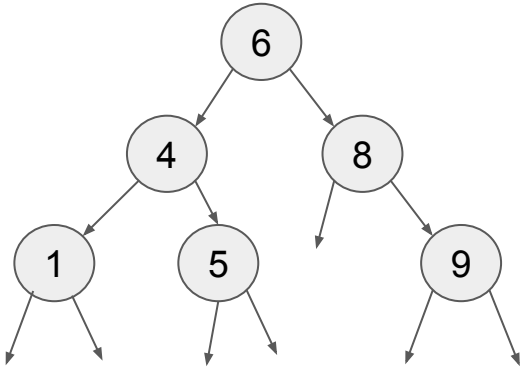
Set using a BST

```
template <typename T>

class BST_Set {

public:

    bool contains(const T &v) const { return elts.contains(v); }

    void insert(const T &v){ if(!contains(v)) elts.insert(); }

    int size() const { return elts.size(); }



private:

    BinarySearchTree<T> elts;

};
```

Set Efficiency

A Set using a BST under the hood?

Function	Unsorted_Set	Sorted_Set	BST_Set
`size`	O(1)	O(1)	
`contains`	O(n)	O(log n)	O(log n)
`insert`	O(n)	O(n)	O(log n)



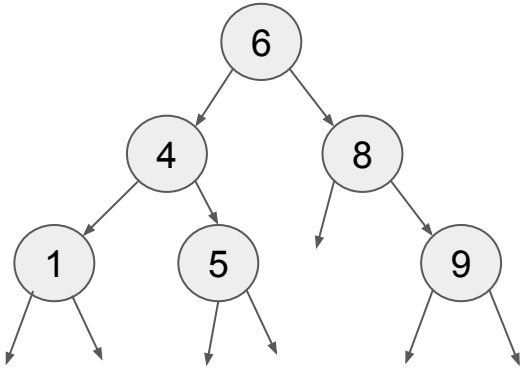
Set using a BST

```
template <typename T>

class BST_Set {

public:

    bool contains(const T &v) const { return elts.contains(v); }

    void insert(const T &v){ if(!contains(v)) elts.insert(); }

    int size() const { return elts.size(); }


private:

    BinarySearchTree<T> elts;

};
```

Set Efficiency

A Set using a BST under the hood?

Function	Unsorted_Set	Sorted_Set	BST_Set
`size`	O(1)	O(1)	O(1)
`contains`	O(n)	O(log n)	O(log n)
`insert`	O(n)	O(n)	O(log n)



Agenda

Functors Review

Containers Review

-BSTs

-Sets

-Maps

C++ Shortcuts

Recursion Practice

Review: Containers - holds elements

- Arrays
- Vectors
- Unordered_Set
- Ordered_Set
- List
- Binary Search Tree (BST)
- **Map**

Has things like:
- Insert (push_front, push_back…)
- Contains
- Size
- Remove
- Iterators

Map

Data structure that associates keys with values

Example: We want to keep track of definitions of words

“Tree” → “Green tall thing outside”

“Candle” → “Smells good and lights on fire”

Map

```
int main(){

    Map<string, string> dictionary;

    dictionary[“Tree”] = “Green tall thing outside”;

    dictionary[“Candle”] = “Smells good and lights on fire”;

}
```

Keys<string>	**Value<string>**
“Tree”	“Green tall thing outside”
“Candle”	“Smells good and lights on fire”

Map

Data structure that associates keys with values

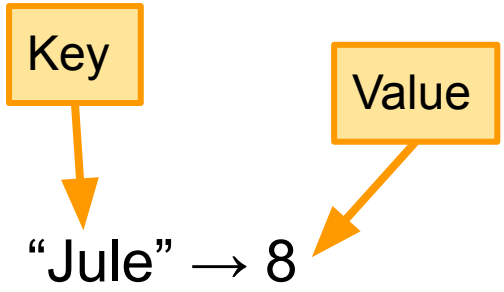
Example: We want to keep track of scores in a friendly neighborhood euchre tournament

“Jule” → 8

“Daniel” → 9

“Sophia” → 11

Map




```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    scores[“Daniel”] = 9;

    scores[“Sophia”] = 11;

}
```

Keys<string>	Value<int>
“Jule”	8
“Daniel”	9
“Sophia”	11

Using a map

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

Using a map

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

Inserts the key “Jule” with the value 8

Using a map

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

Inserts the key “Jule” with the value 8

Changes the value at key “Jule” by 1000

How to represent a Map Under the hood?

- **Fast insertion of a new key value pair**
- Fast access to a value given a key

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

How to represent a Map Under the hood?

- Fast insertion of a new key value pair
- **Fast access to a value given a key**

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

How to represent a Map Under the hood?

- Fast insertion of a new key value pair
- Fast access to a value given a key

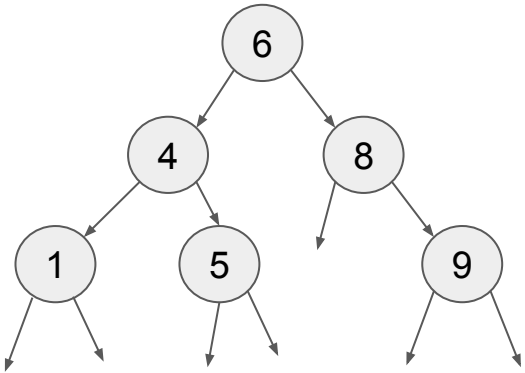
Binary Search Tree!

```
int main(){

    Map<string, int> scores;

    scores[“Jule”] = 8;

    cout << scores[“Jule”] << endl;

    scores[“Jule”] = scores[“Jule”] + 1000;

}
```

Representing a Map using a BST

```
template <typename T>
class BinarySearchTree {
public:
  BinarySearchTree();
  BinarySearchTree(const BinarySearchTree &other);
  BinarySearchTree & operator=(const BinarySearchTree &other);
  ~BinarySearchTree();

  bool empty() const;
  int size() const;
  bool contains(const T &item) const;
  void insert(const T &item);

private:
  struct Node {
    T datum;
    Node *left, *right;
  };

  Node *root;
};
```

What will be type T?



Pair! A struct that holds two values.

std::pair

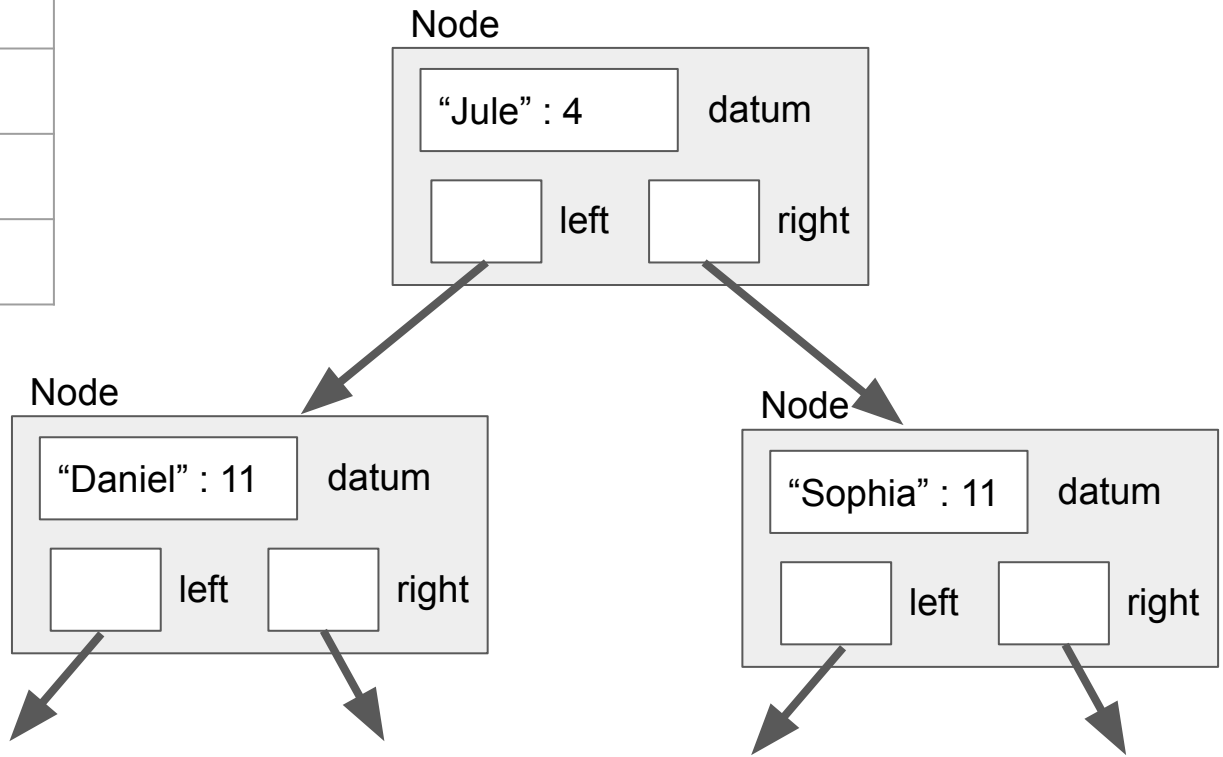
- A templated struct with a member called `first` and another called `second`

```
int main(){

    std::pair<int, bool> p1;

    p1.first = 5;

    p1.second = false;

}
```

Representing a Map using a BST under the hood

Keys<string>	Value<int>
“Jule”	4
“Daniel”	11
“Sophia”	11

typename T = pair<string, int>

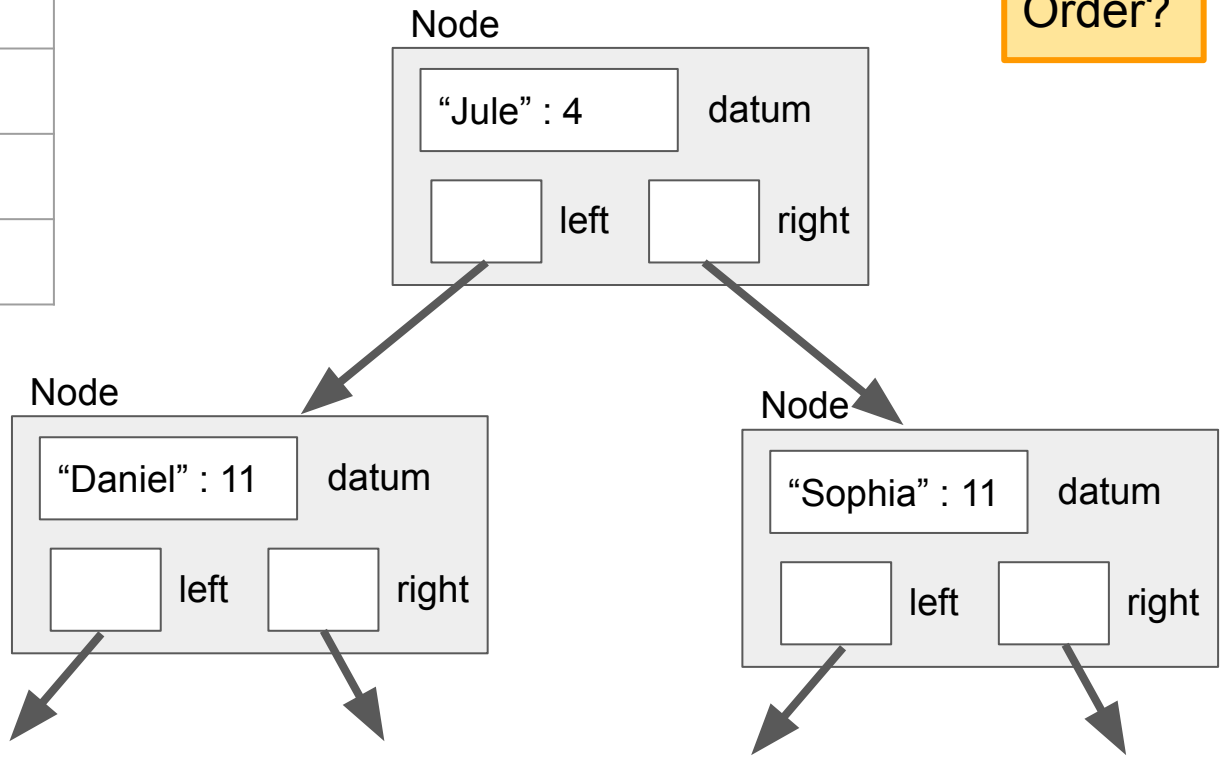


Representing a Map using a BST under the hood

Keys<string>	Value<int>
“Jule”	4
“Daniel”	11
“Sophia”	11

typename T = pair<string, int>

Order?



Problem: Comparing Pairs

- Computer doesn't know how to compare two pairs

```
int main(){

    std::pair<int, double> p1 = {5, 45.2};

    std::pair<int, double> p2 = {50, 6.7};

}
```

Key?
Value?

Problem: Comparing Pairs

- Computer doesn't know how to compare two pairs

```
int main(){

    std::pair<int, double> p1 = {5, 45.2};

    std::pair<int, double> p2 = {50, 6.7};

}
```

Key?
Value?

Map - sorted based on key

Map using a BST

Making a BST work with pairs

1. Change BST to use a custom comparator functor when comparing two elements in the tree

   ex) insert

2. Create a custom comparator functor for the `std::pair` used in a map to then use in the underlying BST

Map using a BST

Making a BST work with pairs

1. **Change BST to use a custom comparator functor when comparing two elements in the tree**

   ex) insert

2. Create a custom comparator functor for the `std::pair` used in a map to then use in the underlying BST

BST with custom comparator

```
template <typename T, typename Compare=std::less<T>>
class BinarySeachTree {
    …
private:
    Node* root;
    Compare less;
}
```

BST with custom comparator

```
template <typename T, typename Compare=std::less<T>>
class BinarySeachTree {
    …
private:
    Node* root;
    Compare less;
}
```

1. Create a template for the type of the comparator

BST with custom comparator

```
template <typename T, typename Compare=std::less<T>>
class BinarySeachTree {
    ...
private:
    Node* root;
    Compare less;
}
```

1. Create a template for the type of the comparator
2. Create an object of that templated type

BST with custom comparator

```
template <typename T, typename Compare=std::less<T>>
class BinarySeachTree {
    ...
private:
    Node* root;
    Compare less;
}
```

1. Create a template for the type of the comparator
2. Create an object of that templated type

Can now use on two objects of type T with less to determine which is less than the other!

```
cout << less(a, b) << endl;
```

BST with custom comparator

```
template <typename T,

typename Compare=std::less<T>>

class BinarySeachTree {

    …

private:

    Node* root;

    Compare less;

}
```

```
class DuckNameLess {

public:

    bool operator()(const Duck &d1, const
Duck &d2) const{ return d1.getName() <
d2.getName(); }

};
```

```
int main(){

    BinarySearchTree<Duck, DuckNameLess> bst;

}
```

BST with custom comparator

```
template <typename T,

typename Compare=std::less<T>>

class BinarySeachTree {

    …

private:

    Node* root;

    Compare less;

}
```

```
class DuckNameLess {

public:

    bool operator()(const Duck &d1, const
Duck &d2) const{ return d1.getName() <
d2.getName(); }

};
```

Type of the comparator!

```
int main(){

    BinarySearchTree<Duck, DuckNameLess> bst;

}
```

BST with custom comparator

```
template <typename T,

typename Compare=std::less<T>>

class BinarySeachTree {

    …

private:

    Node* root;

    Compare less;

}
```

Actual object of type DuckNameLess

```
class DuckNameLess {

public:

    bool operator()(const Duck &d1, const
Duck &d2) const{ return d1.getName() <
d2.getName(); }

};
```

Type of the comparator!

```
int main(){

    BinarySearchTree<Duck, DuckNameLess> bst;

}
```

Making a Map using a BST

1. Change BST to use a custom comparator functor when comparing two elements in the tree

   ex) insert

2. **Create a custom comparator functor for the `std::pair` used in a map to then use in the underlying BST**

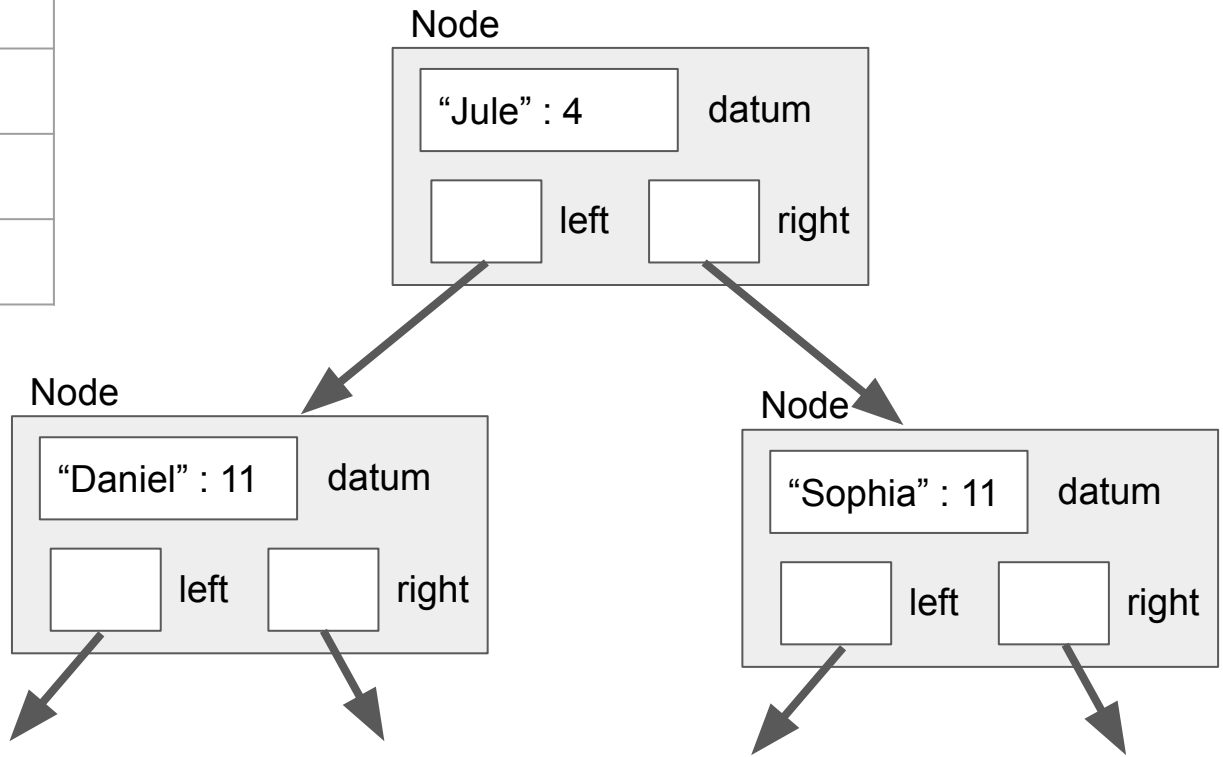
Sorted by key! AKA first thing in the pair

Representing a map using a BST

Pairs Sorted by key

Keys<string>	Value<int>
“Jule”	4
“Daniel”	11
“Sophia”	11

typename T = pair<string, int>



Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>
class Map {
    …
private:
    using Pair_type = std::pair<Key_type, Value_type>;




    BinarySearchTree<Pair_type, ???? > entries;
}
```

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>

class Map {

    …

private:

    using Pair_type = std::pair<Key_type, Value_type>;

    BinarySearchTree<Pair_type, ???? > entries;

}
```

Key compare takes in two values of type Key_type and tells you which is less

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>
class Map {
    ...
private:
    using Pair_type = std::pair<Key_type, Value_type>;

    BinarySearchTree<Pair_type, ???? > entries;
}
```

Key compare takes in two values of type Key_type and tells you which is less

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>
class Map {
    ...
private:
    using Pair_type = std::pair<Key_type, Value_type>;

    BinarySearchTree<Pair_type, ???? > entries;
}
```

Key compare takes in two values of type Key_type and tells you which is less

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>
class Map {
    ...
private:
    using Pair_type = std::pair<Key_type, Value_type>;
    BinarySearchTree<Pair_type, ???? > entries;
}
```

Key compare takes in two values of type Key_type and tells you which is less

Could we pass in Key_compare?

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>
class Map {
    ...
private:
    using Pair_type = std::pair<Key_type, Value_type>;

    BinarySearchTree<Pair_type, ???? > entries;
}
```

Key compare takes in two values of type Key_type and tells you which is less

Need a functor that can compare pairs given a the functor type Key_compare!

Implementing a Map

```
template <typename Key_type, typename Value_type, typename Key_compare>

class Map {

     …

private:

     using Pair_type = std::pair<Key_type, Value_type>;

     class PairComp {

          public:

               bool operator(){....}

     };

     BinarySearchTree<Pair_type, PairComp> entries;

}
```

Need a functor that can compare pairs given a the functor type Key_compare!

Agenda

Functors Review

Containers Review

- -BSTs
- -Sets
- -Maps

C++ Shortcuts

Recursion Practice

Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks


    //loop through ducks?




}
```

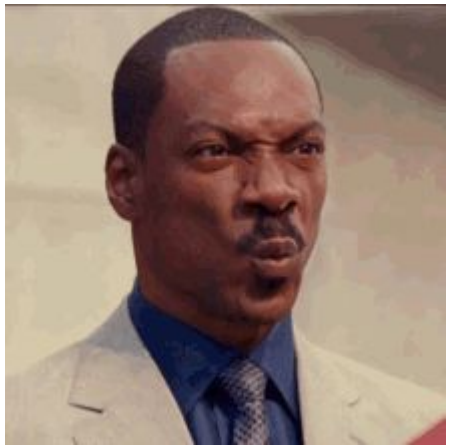
Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks


    //loop through ducks?

}
```

Iterators!

Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks

    for(typename Map<Duck, int, DuckNameLess>::Iterator it = ducks.begin();
it != ducks.end(); ++it){

        cout << it->first.getName() << endl;

        cout << it->second << endl;

    }

}
```



Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks

    for(auto it = ducks.begin(); it != ducks.end(); ++it){

        cout << it->first.getName() << endl;

        cout << it->second << endl;

    }

}
```

C++ Shortcuts

Auto - Compiler deduces the type for you

Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks

    for(auto it = ducks.begin(); it != ducks.end(); ++it){

        cout << it->first.getName() << endl;

        cout << it->second << endl;

    }

}
```

Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks

    for(auto item : ducks){

        cout << item.first.getName() << endl;

        cout << item.second << endl;

    }

}
```

C++ Simplified

Auto - Compiler deduces the type for you

Range-Based For Loops - loops through a container using an iterator

Range-Based For Loops

```
vector<int> vec = {1, 2, 3};

for(int item : vec) {

cout << item << endl;

}
```

item = *iterator = int

```
for (auto it = vec.begin(); it != vec.end(); ++it) {

  int item = *it;

  cout << item << endl;

}
```

Range-Based For Loops

```
vector<int> vec = {1, 2, 3};

for(int item : vec) {

cout << item << endl;

}
```

item = *iterator = int

What if we want to change elements in the vector?

```
for (auto it = vec.begin(); it != vec.end(); ++it) {

  int item = *it;

  cout << item << endl;

}
```

If we want to change the items in vector

```
vector<int> vec = {1, 2, 3};

for(int &item : vec) {

 item = 42;

}
```

item = *iterator = int

```
for (auto it = vec.begin(); it != vec.end(); ++it) {

   int &item = *it;

   item = 42;

}
```

Using a map…

```
int main(){

    Map<Duck, int, DuckNameLess> ducks; //insert ducks

    for(auto item : ducks){

        cout << item.first.getName() << endl;

        cout << item.second << endl;

    }

}
```

item = *iterator = pair<duck, int>

Agenda

Functors Review

Containers Review

- -BSTs
- -Sets
- -Maps

C++ Shortcuts

Recursion Practice

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

Can you break this up into smaller pieces?

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

Can you break this up into smaller pieces?

```
void reverse(int *left, int *right){ … }
```

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

Can you break this up into smaller pieces?

```
void reverse(int *left, int *right){ … }
```

If you switch the first and last element, what is left to do?

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

```
void reverse(int *left, int *right){

      //base case

      int temp = *left;

      *left = *right;

      *right = temp;

      //recursive call

}
```

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

```
void reverse(int *left, int *right){

      //base case

      int temp = *left;

      *left = *right;

      *right = temp;

      reverse(left + 1, right - 1); //recursive call

}
```

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

```
void reverse(int *left, int *right){

      if(left >= right) return; //base case

      int temp = *left;

      *left = *right;

      *right = temp;

      reverse(left + 1, right - 1); //recursive call

}
```

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

```
void reverse(int *left, int *right){

      if(left >= right) return; //base case

      int temp = *left;

      *left = *right;

      *right = temp;

      reverse(left + 1, right - 1); //recursive call

}
```

Tail recursive?

Writing Recursion

https://onlinegdb.com/Lq-V3U2U0

Reverse an array using recursion!

1	2	3	4	5

```
void reverse(int *left, int *right){

      if(left >= right) return; //base case

      int temp = *left;

      *left = *right;

      *right = temp;

      reverse(left + 1, right - 1); //recursive call

}
```

Tail recursive?
YES!

Exercise: Fibonacci

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Write a fibonacci function using iteration
Write a fibonacci function using recursion
Write a fibonacci function using tail-recursion

https://onlinegdb.com/nEAU_vo6E